* Graph traversals
  + Node traversal is to visit every node in a graph (usually what this refers to)
  + Edge traversal is to visit every edge in a graph
* Application of node traversal
  + Get/display data store at all nodes, get node count
  + Search nodes for data and processing
  + To derive graph property like connectivity
* Applications of edge traversal
  + Get/display data (e.g. Weight) stored in all edges, get edge count, get total edge weight, search edges for certain info

**Graph traversal/search algorithms**

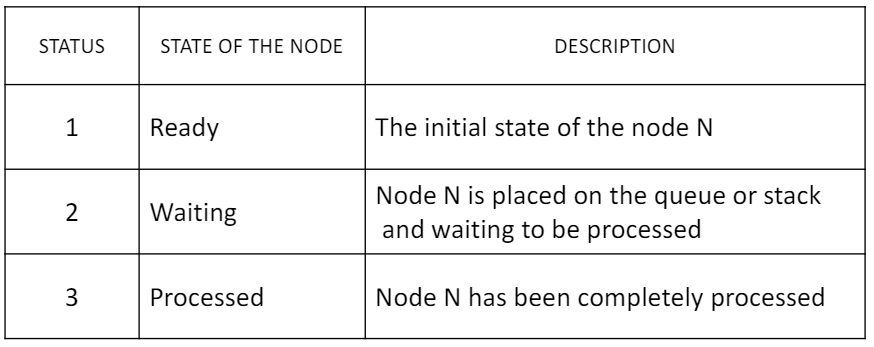
* Graph traversal algorithms are usually designed for applications and depend on the representation for graphs
  + Node traversal by node list
  + Edge traversal by edge list
  + Node/edge traversal by node-edge relations
* Graph traversal algorithms
  + Breadth first traversal
  + Depth first traversal
* Graph search algorithms (a variation of traversal)
  + BFS
  + DFS
  + A\* search
  + Heuristic search

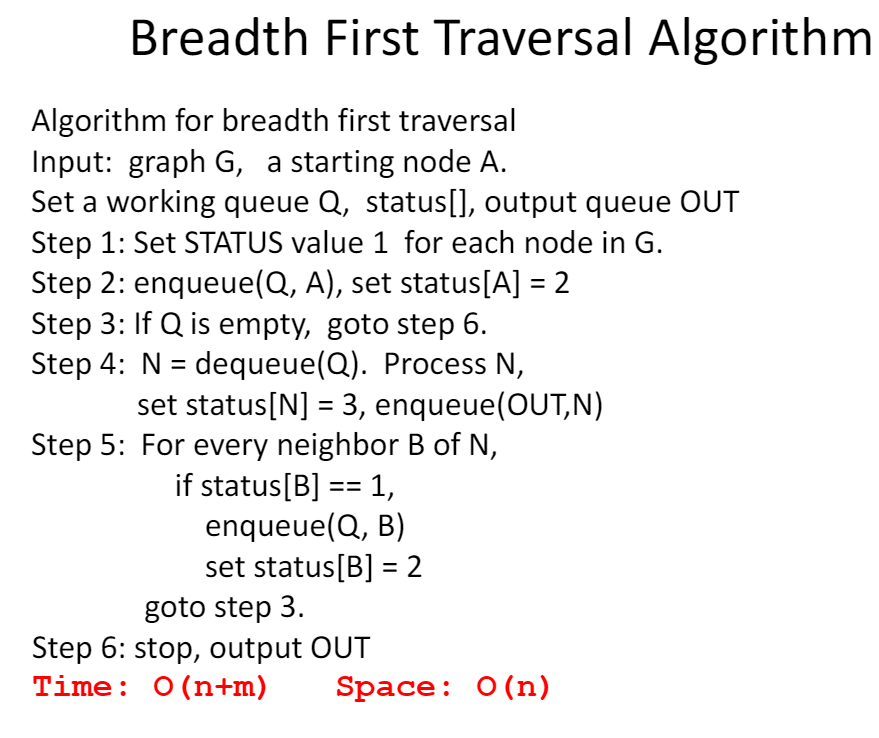
**Breadth first traversal algorithm**

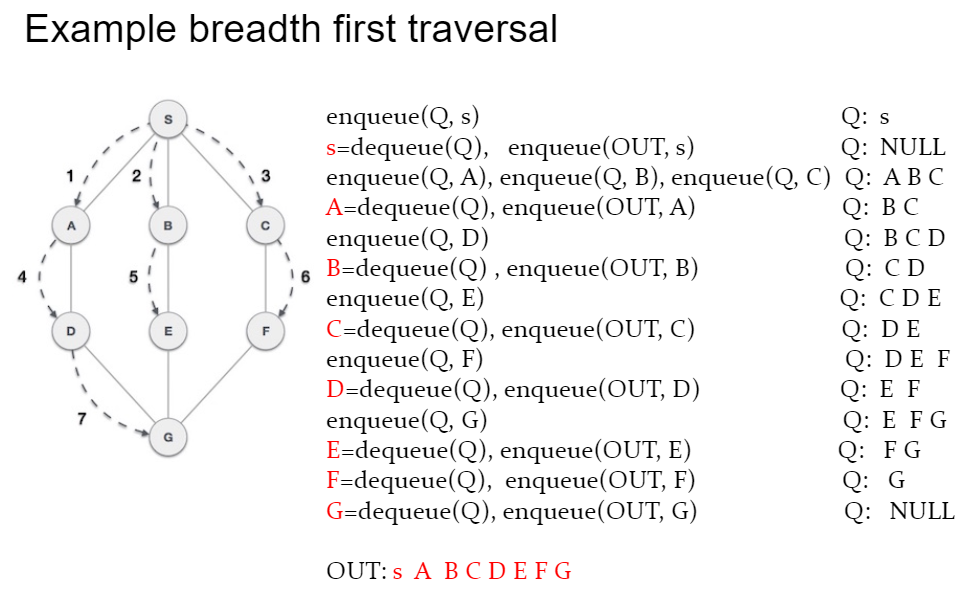
* Breadth-first traversal algorithm beings at a root node and explores all the neighboring nodes. Then for each of those neighbor nodes, the algorithm explores their unexplored neighbor nodes, and so on, until all nodes are explored.
* Another view: starting from node A, for I from 1 to n, visits all nodes at distance I from A one after another, increase I by 1.
* This algorithm only works for connected graphs, for unconnected graphs, it needs to traverse each component.
* Similar to breadth first traversal of trees, breadth first traversal of a graph needs a queue data structure to remember the ordered front line of traversal. At a node, it enqueues unvisited nodes.

**How to remember if a node is visited or not?**

* Generally, we can use a data structure (linked list, BST, AVL, hash table) to store visited nodes
* A simple method is to use an int array to represent the status of nodes. During the execution of the algorithm, every node in the graph will have a status value. For example:

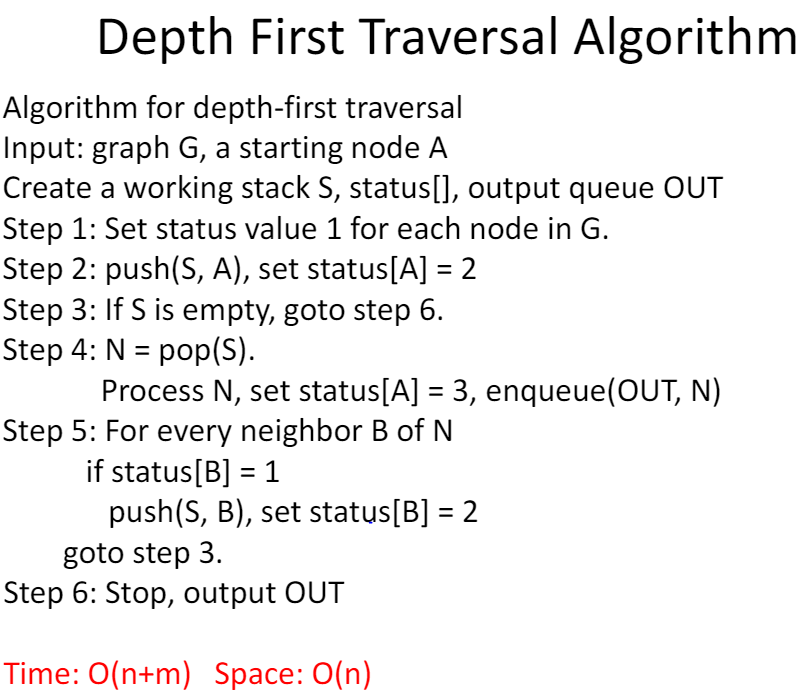


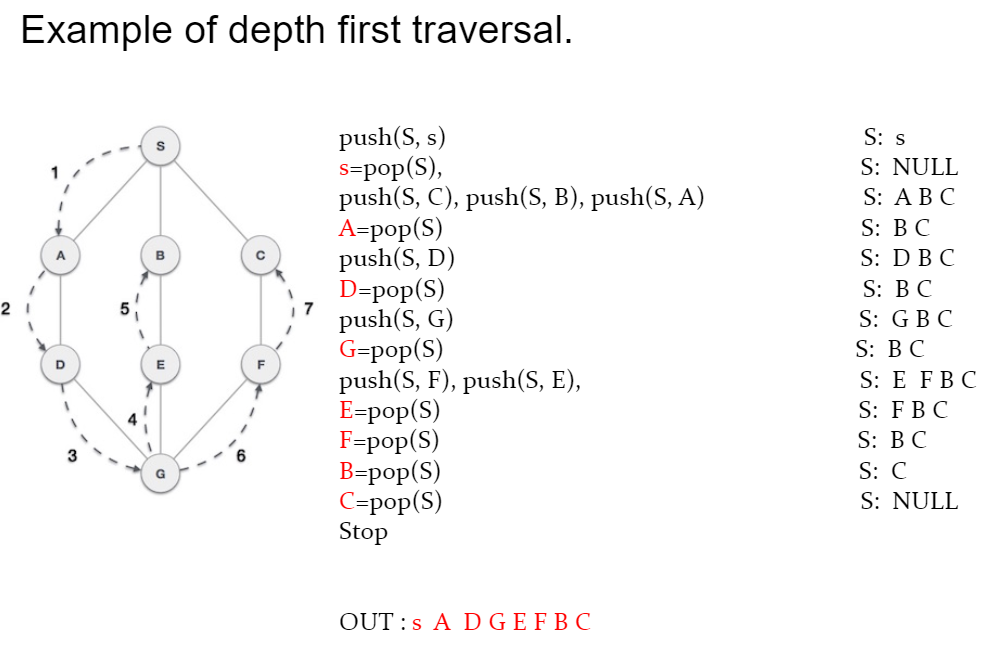




**Depth first traversal**

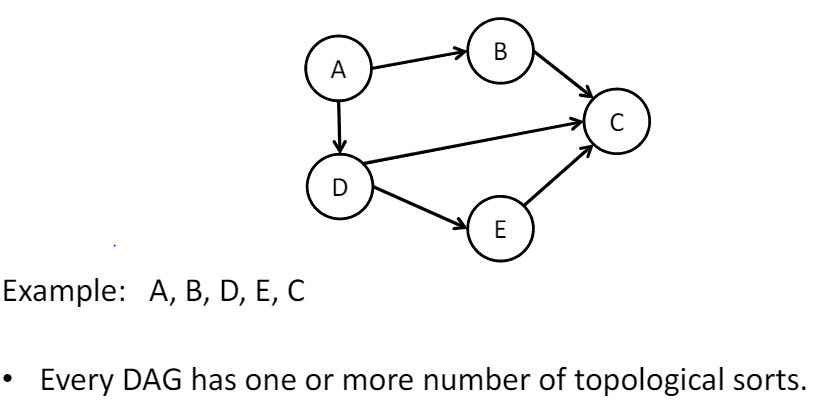
* The depth first traversal starts from a node of G and explores a unexplored node (deeper and deeper) until a node that has no neighbor being in ready state. When a dead-end is reached, the algorithm backtracks, returning to the most recent node that has not been completely explored.
* Similar to depth first traversal of trees, depth first traversal of a graph needs a stack data structure to remember the path for backtracking and next node to explore. It pops a node from the stack, processes it, and then pushes its’ unvisited neighbors to the stack.





**Topological sorting**

* Topological sort of a graph is a traversal of a directed acyclic graph (DAG) G in a linear ordering or nodes of G such that if (u,v) is an edge of G, then u appears before v in the ordering.



**Application of topologic sorting**

* Topologic sorting is used in scheduling applications
  + The tasks that have to be completed are represented by nodes and there is an edge from node u to v id task u must be completed before task v can begin.
  + A topological sort gives us a scheduling order of tasks
* Topological sorting is used in software component dependency management
  + For example, when a C project consists many component files, .h .c programs, the make build tool needs to build each component on order so that a later component can be built if it uses a previously included component.

